Faculdade de Engenharia da Universidade do Porto

LinkedIn

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

4º ano

Métodos Formais de Engenharia de Software

Prof. Ana Cristina Ramada Paiva

Prof. João Carlos Pascoal Faria

Estudantes & Autores:

António Ramadas -up201303568 - antonio.ramadas@fe.up.pt

Duarte Pinto - up201304777 - up201304777@fe.up.pt

Marina Camilo - 201307722 - up201307722@fe.up.pt

8 de Janeiro de 2017

Índice

1. Descrição do Sistema

1.1. Descrição informal do sistema

Neste projecto pretende-se modelar a rede social LinkedIn. Nesta rede é possível:

- Determinar a distância entre duas pessoas
- Colocar CV's
- Procurar pessoas
- Calcular os contactos comuns entre duas pessoas
- Determinar a pessoa com mais contactos
- Calcular a distância média entre pessoas na rede
- Criar grupos de utilizadores
- Mandar mensagens entre grupos

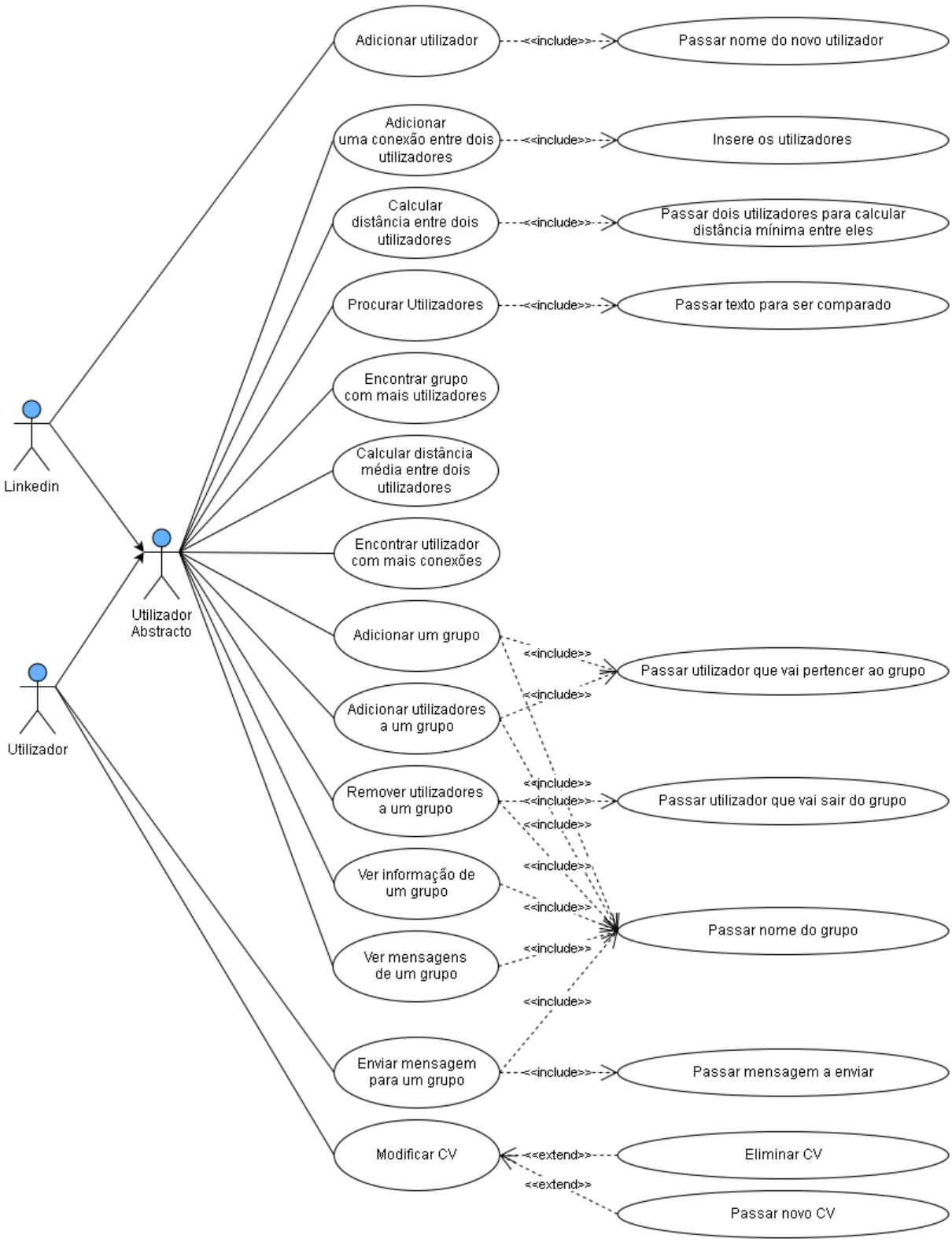
1.2. Lista de requisitos

Para cumprir o objectivo são necessários os seguintes requisitos:

Requisitos	Prioridade	Descrição
R01	Obrigatória	Adicionar utilizador
R02	Obrigatória	Obter ou atualizar o CV
R03	Obrigatória	Adicionar ligações entre utilizadores
R04	Obrigatória	Listar ligações comuns entre utilizadores
R05	Obrigatória	Calcular distância entre dois utilizadores
R06	Obrigatória	Calcular a distância média entre um utilizador e todos os restantes utilizadores
R07	Obrigatória	Calcular a distância média entre todos os utilizadores
R08	Obrigatória	Obter o utilizador mais conhecido (com mais conexões)
R09	Obrigatória	Pesquisar utilizadores pelo nome com ordenação
R10	Opcional	Adicionar grupo
R11	Opcional	Listar nome do grupos
R12	Opcional	Obter a informação de um grupo
R13	Opcional	Adicionar e remover utilizador a um grupo
R14	Opcional	Enviar mensagem para o grupo
R15	Opcional	Listar mensagens de um grupo
R16	Opcional	Pesquisar o grupo mais popular (com mais utilizadores)

2. UML

2.1. Use Case Model



Cenário	Configuração
Descrição	Adicionar utilizadores à rede Linkedin.
Pré-condições	1. O utilizador não pode pertencer à rede Linkedin. (input)
Pós-condições	1. O Utilizador adicionado tem de pertencer à rede Linkedin. (final system state)
Passos	1. No menu “Users” o administrador escolhe a opção “Add User”. 2. O administrador passa o utilizador que quer adicionar. 3. O adminstrador volta ao menu anterior.
Exceções	1. O utilizador já se encontra na rede. (passo 3)

Cenário	Configuração
Descrição	Adicionar uma conexão entre dois utilizadores da rede Linkedin.
Pré-condições	1. Os utilizadores são diferentes. (input) 2. Ambos os utilizadores pertencem à rede Linkedin. (input) 3. Não existe previamente uma conexão entre os utilizadores. (input)
Pós-condições	1. A conexão final tem de ser bilateral. (final system state)
Passos	1. No menu “Users” o administrador escolhe a opção “Add a new connection between two users”. 2. O administrador passa os utilizadores que pretende conectar. 3. O adminstrador volta ao menu anterior.
Exceções	1. Os utilizadores ou um deles não pertence à rede Linkedin. (passo 3) 2. Os utilizadores passados são a mesma pessoa. (passo 3) 3. Os utilizadores já tem uma conexão entre si. (passo 3)

Cenário	Configuração
Descrição	Calcular distância entre dois utilizadores
Pré-condições	1. Os utilizadores têm de estar associados à rede Linkedin. (input)
Pós-condições	(none)
Passos	1. No menu “Users” o administrador escolhe a opção “Distance between two users”. 2. O administrador passa os utilizadores. 3. A rede Linkedin devolve a distância entre os utilizadores.
Exceções	1. Os utilizadores ou utilizador não se encontra na rede. (passo 3)

Cenário	Configuração
Descrição	Procurar utilizadores
Pré-condições	1. O nome do utilizador tem de ter comprimento maior que zero. (input)
Pós-condições	(none)
Passos	1. O administrador escolhe a opção “Search Users by name”. 2. O administrador passa o nome de utilizador que quer procurar. 3. A rede Linkedin devolve os utilizadores com nome similar.
Exceções	1. O comprimento do nome é menor ou igual a zero. (passo 3)

Cenário	Configuração
Descrição	Encontrar o grupo na rede Linkedin com mais utilizadores.
Pré-condições	(none)
Pós-condições	1. O nome do grupo tem de representar o grupo com mais utilizadores na rede Linkedin. (output)
Passos	1. No menu “Stats” o administrador escolhe a opção “Most famous group”. 2. A rede Linkedin devolve o nome do grupo com mais utilizadores.
Exceções	(none)

Cenário	Configuração
Descrição	Calcular distância média entre dois utilizadores da rede Linkedin.
Pré-condições	1. A rede Linkedin tem de ter utilizadores. (initial system state)
Pós-condições	(none)
Passos	1. No menu “Stats” o administrador escolhe a opção “Average User Distance”. 2. A rede Linkedin devolve a distância média entre dois utilizadores.
Exceções	1. A rede Linkedin não tem utilizadores. (passo 2)

Cenário	Configuração
Descrição	Encontrar utilizador com mais conexões na rede Linkedin.
Pré-condições	1. A rede Linkedin tem de ter utilizadores. (initial system state)
Pós-condições	1. O utilizador tem de ter o maior número de conexões da rede. (output)
Passos	1. No menu “Stats” o administrador escolhe a opção “Most famous user”. 2. A rede Linkedin devolve o utilizador com mais conexões.
Exceções	1. A rede Linkedin não tem utilizadores. (passo 2)

Cenário	Configuração
Descrição	Adicionar um grupo à rede Linkedin
Pré-condições	1. O nome do grupo tem de ser diferente de todos os nomes dos grupos que a rede Linkedin já possua. (input)
Pós-condições	(none)
Passos	1. No menu “Groups” o administrador escolhe a opção “Add a new group”. 2. O administrador passa o nome do grupo. 3. O adminstrador volta ao menu anterior.
Exceções	1. O nome do grupo já existe num grupo da rede Linkedin. (passo 3)

Cenário	Configuração
Descrição	Adicionar utilizador a um grupo da rede Linkedin
Pré-condições	1. O nome do grupo tem de representar um grupo da rede Linkedin. (input) 2. O utilizador tem de estar na rede Linkedin. (input)
Pós-condições	(none)
Passos	1. No menu “Groups” o administrador escolhe a opção “See groups’s info”. 2. O administrador passa nome do grupo. 3. O administrador escolhe a opção “Add existing user to group”. 4. O administrador passa o nome de utilizador e o nome do grupo. 5. O adminstrador volta ao menu anterior.
Exceções	1. O nome do grupo não representa um grupo. (passo 3)

	2. O utilizador não pertence à rede Linkedin (passo 5)

Cenário	**Configuração**
Descrição	Remover utilizador a um grupo da rede Linkedin
Pré-condições	1. O nome do grupo tem de representar um grupo da rede Linkedin. (input) 2. O utilizador tem de estar na rede Linkedin. (input)
Pós-condições	(none)
Passos	1. No menu “Groups” o administrador escolhe a opção “See groups’s info”. 2. O administrador passa nome do grupo. 3. O administrador escolhe a opção “Remove user from group”. 4. O administrador passa o nome de utilizador. 5. O adminstrador volta ao menu anterior.
Exceções	1. O nome do grupo não representa um grupo. (passo 3) 2. O utilizador não pertence à rede Linkedin (passo 5)

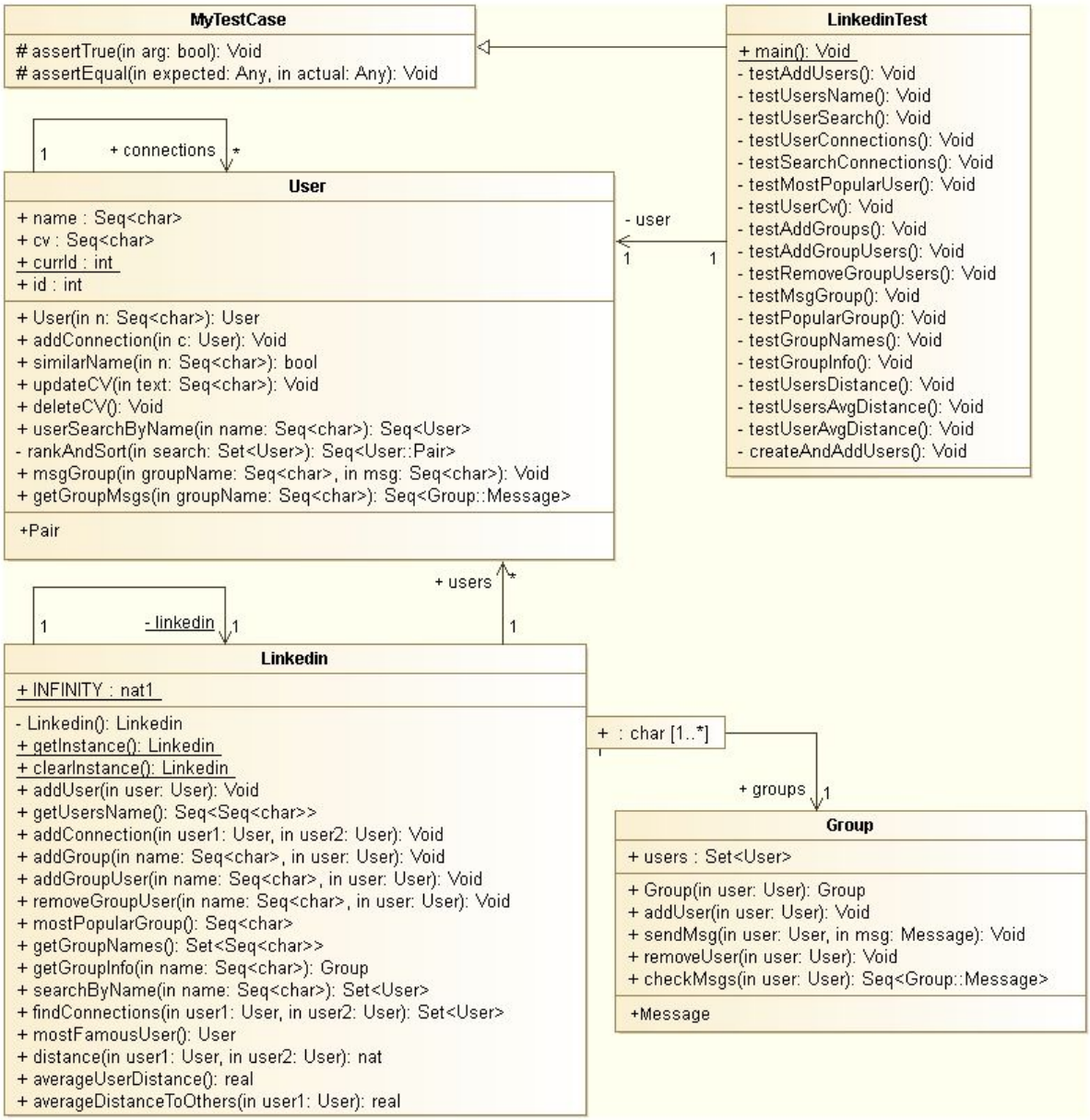
Cenário	**Configuração**
Descrição	Ver informação de um grupo da rede Linkedin
Pré-condições	1. O nome do grupo tem de representar um grupo da rede Linkedin. (input)
Pós-condições	(none)
Passos	1. No menu “Groups” o administrador escolhe a opção “See groups’s info”. 3. O administrador passa o nome do grupo. 4. O Linkedin apresenta a informação do grupo.
Exceções	1. O nome do grupo não representa um grupo. (passo 4)

Cenário	Configuração
Descrição	Ver mensagens de um grupo da rede Linkedin
Pré-condições	1. O nome do grupo tem de representar um grupo da rede Linkedin. (input) 2. O utilizador tem de estar na rede Linkedin. (input)
Pós-condições	(none)
Passos	1. No menu “Users” o administrador escolhe a opção “See User’s Profile”. 2. O administrador escolhe um utilizador. 3. O administrador escolhe a opção “Receive messages from a group” 4. O administrador passa o nome do grupo 5. O Linkedin apresenta as mensagens do grupo.
Exceções	1. O utilizador não pertence à rede Linkedin (passo 2) 2. O nome do grupo não representa um grupo. (passo 5)

Cenário	Configuração
Descrição	Modificar CV
Pré-condições	1. O CV não pode ser indefinido. (input)
Pós-condições	1. O CV final do utilizador tem de ser igual ao input. (final system state)
Passos	1. No menu “Users” o administrador escolhe a opção “See User’s Profile”. 2. O administrador escolhe um utilizador. 3. O administrador escolhe a opção “Update CV” 4. O adminstrador volta ao menu anterior.
Exceções	1. O CV é indefinido. (passo 4)

Cenário	Configuração
Descrição	Eliminar CV
Pré-condições	(none)
Pós-condições	1. O CV final do utilizador tem de ser uma string vazia. (final system state)
Passos	1. No menu “Users” o administrador escolhe a opção “See User’s Profile”. 2. O administrador escolhe um utilizador. 3. O administrador escolhe a opção “Eliminar CV” 4. O adminstrador volta ao menu anterior.
Exceções	(none)

2.2. Class Model



Classe	Descrição
Group	Define os grupos de utilizadores que podem ser criados no Linkedin, tal como as operações que lhes podem ser aplicados.
Linkedin	Core model; define a rede Linkedin e as operações que se podem realizar na mesma.
User	Define os utilizadores que pertencem à rede Linkedin
MyTestCase	Superclasse para testar classes; define assertEquals e assertTrue.
LinkedinTest	Define os cenários de teste/uso e testa casos para a rede Linkedin.

3. Modelo Formal VDM++

3.1. Classe Group

```
class Group
types
	public Message = seq1 of char;
values
-- TODO Define values here
instance variables
	public users: set of User := {};
	public msgs: seq of Message := [];

	-- must be present at least one user in the group
	inv card users >= 1;

operations
	public Group: User ==> Group
	Group(user) == (
		users := {user};
		return self
	)
	pre user in set Linkedin`getInstance().users
	post users = {user} and msgs = [];


	public addUser: User ==> ()
	addUser(user) == users := users union {user}
	pre user not in set users
	post users = users~ union {user};


	public sendMsg: User * Message ==> ()
	sendMsg(user, msg) == msgs := msgs ^ [msg]
	pre user in set users
	post msgs = msgs~ ^ [msg];


	public removeUser: User ==> ()
	removeUser(user) == users := users \ {user}
	pre user in set users and
			card users > 1
	post users = users~ \ {user};


	public pure checkMsgs: User ==> seq of Message
	checkMsgs(user) == return msgs
	pre user in set users;
```

```
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Group
```

Function or operation	Line	Coverage	Calls
Group	14	100.0%	11
addUser	23	100.0%	6
checkMsgs	42	100.0%	1
removeUser	35	100.0%	1
sendMsg	29	100.0%	1
Group.vdmpp		100.0%	20

3.2. Classe Linkedin

```
class Linkedin
types
-- TODO Define types here
values
	public INFINITY: nat1 = 9999;

instance variables
	public users: set of User := {};

	public groups: map seq1 of char to Group := {|->};

	private static linkedin: Linkedin := new Linkedin();

	-- invariant to maintain the correctness (each connection is bidirectional)
	inv forall user1 in set users &
				forall user2 in set user1.connections &
					user1 in set user2.connections;

operations
	private Linkedin: () ==> Linkedin
	Linkedin() == return self
	post users = {};


	-- singleton - return the existent instance
	public pure static getInstance: () ==> Linkedin
	getInstance() == return linkedin;
```

```
-- singleton - reset the instance
public static clearInstance: () ==> Linkedin
clearInstance() == (
	linkedin := new Linkedin();
	return getInstance();
)
post RESULT.users = {};


public addUser: User ==> ()
addUser(user) == users := users union {user}
pre user not in set users
post users = users~ union {user};


public getUsersName: () ==> seq of seq of char
getUsersName() == (
	dcl names: seq of seq of char := [];

	for all user in set users do
		names := names ^ [user.name];

	return names
);


public addConnection: User * User ==> ()
addConnection(user1, user2) == (
--|| (user1.addConnection(user2), user2.addConnection(user1))
	dcl user1_temp: User := user1;
	dcl user2_temp: User := user2;
	atomic (
		user1_temp.connections := user1.connections union {user2};
		user2_temp.connections := user2.connections union {user1}
	);
)
pre
		-- they are not the same user
		user1 <> user2 and
		-- both users exist in the database
		user1 in set users and
		user2 in set users and
		-- there's no connection between them
		user2 not in set user1.connections and
		user1 not in set user2.connections
post user2 in set user1.connections and
		user1 in set user2.connections;


public addGroup: seq1 of char * User ==> ()
```

```
	addGroup(name, user) == groups := groups munion {name |-> new Group(user)}
	pre
			-- each group name is unique
			name not in set dom groups and
			user in set users;


	public addGroupUser: seq1 of char * User ==> ()
	addGroupUser(name, user) == groups(name).addUser(user)
	pre user in set users and
			name in set dom groups;


	public removeGroupUser: seq1 of char * User ==> ()
	removeGroupUser(name, user) == groups(name).removeUser(user)
	pre user in set users and
			name in set dom groups;


	public pure mostPopularGroup: () ==> seq1 of char
	mostPopularGroup() == (
		dcl name: [seq1 of char] := nil;

		for all groupName in set dom groups do
			if name = nil or card groups(groupName).users > card
groups(name).users then
				name := groupName;

		return name
	)
	post RESULT in set dom groups and
			card {g | g in set dom groups & card groups(g).users > card
groups(RESULT).users} = 0;


	public pure getGroupNames: () ==> set of seq1 of char
	getGroupNames() == return dom groups;


	public pure getGroupInfo: seq1 of char ==> Group
	getGroupInfo(name) == return groups(name)
	pre name in set dom groups;


	-- returns all users that have a similar name to the given one
	public pure searchByName: seq of char ==> set of User
	searchByName(name) == return {user | user in set users &
user.similarName(name)} --(
	--dcl results: set of User := {};

	--for all user in set users do (
```

```
	--	if user.similarName(name) then
	--		results := results union {user};
	--);

	--return results)
	pre len name > 0;


	-- returns commun contacts between two users
	public pure findConnections: User * User ==> set of User
	findConnections(user1, user2) ==
		return user1.connections inter user2.connections
	pre user1 <> user2 and
		user1 in set users and
		user2 in set users;


	public pure mostFamousUser: () ==> User
	mostFamousUser() ==
	(
		dcl famous: [User] := nil;

		for all user in set users do
			if famous = nil or card user.connections > card
famous.connections then
				famous := user;

		return famous
	)
	pre card users > 0
	post RESULT in set users and
		card {u | u in set users & card u.connections > card
RESULT.connections} = 0;


	-- to calculate the minimum distance between users it is used the breadth
first search algorithm
	public pure distance: User * User ==> nat
	distance(user1, user2) == (
		dcl search: set of User := user1.connections;
		dcl searchTemp: set of User := search;
		dcl counter: nat1 := 1;

		if user1 = user2 then
			return 0;

		while user2 not in set search do (
			counter := counter + 1;

			-- add to the search set the connections of the current users
(add to the search the next level of the tree)
```

```
                for all user in set search do
                        search := search union user.connections;

                -- to prevent an infinite loop
                if card searchTemp = card search then
                        return INFINITY
                else
                        searchTemp := search
        );

        -- if the two users are not reachable then it is returned INFINITY
        if user2 not in set search then
                return INFINITY
        else
                return counter
)
pre user1 in set users and
                user2 in set users;


public pure averageUserDistance: () ==> real
averageUserDistance() == (
        dcl sum: real := 0;
        dcl subSearch: set of User := users;

        for all user1 in set users do (
                subSearch := subSearch \ {user1};
                for all user2 in set subSearch do
                        sum := sum + distance(user1, user2);
        );

        return sum / card users;
)
pre card users > 0
post RESULT >= 0;


public pure averageDistanceToOthers: User ==> real
averageDistanceToOthers(user1) == (
        dcl sum: real := 0;

        for all user2 in set users do
                        sum := sum + distance(user1, user2);

        return sum / (card users - 1);
)
pre card users > 1 and user1 in set users
post RESULT >= 0;
```

functions

```
-- TODO Define functiones here
```

```
traces
-- TODO Define Combinatorial Test Traces here
end Linkedin
```

Function or operation	Line	Coverage	Calls
Linkedin	20	100.0%	18
addConnection	56	100.0%	21
addGroup	79	100.0%	11
addGroupUser	87	100.0%	6
addUser	39	100.0%	141
averageDistanceToOthers	212	100.0%	1
averageUserDistance	195	100.0%	1
clearInstance	31	100.0%	17
distance	162	96.1%	18
findConnections	137	100.0%	1
getGroupInfo	117	100.0%	1
getGroupNames	113	100.0%	1
getInstance	26	100.0%	189
getUsersName	45	100.0%	1
mostFamousUser	145	97.5%	1
mostPopularGroup	99	97.7%	1
removeGroupUser	93	100.0%	1
searchByName	123	100.0%	5
Linkedin.vdmpp		98.9%	435

3.3. Classe User

```
class User
types
	-- simulates a pair
	Pair:: key: nat1 value: User;
values
-- TODO Define values here
instance variables
	public connections: set of User := {};
	public name: seq1 of char;
	public cv: seq of char := "";
	public static currId : int := 0;
	public id : int := currId;

operations
	public User: seq1 of char ==> User
```

```
User(n) == (
	name := n;
	currId := currId +1;
	return self
)
post connections = {} and
		name = n and
		id = currId~ and
		currId = currId~ + 1;


public addConnection: User ==> ()
addConnection(c) == connections := connections union {c}
pre c <> self and c not in set connections
post connections = connections~ union {c};


public pure similarName: seq of char ==> bool
similarName(n) == (
	dcl nameS: seq of char := name;
	dcl found: bool := false;

	while len nameS >= len n and not found do (
		found := true;

		for index = 1 to len n do
			if found and n(index) <> nameS(index) then (
				found := false;
			);

		if found then
			return true
		else (
			nameS := tl nameS;
			found := false;
		);
	);

	return false;
)
pre len n > 0;


public updateCV: seq of char ==> ()
updateCV(text) ==
	cv := text
pre text <> undefined
post cv = text;


public deleteCV: () ==> ()
```

```
		deleteCV() == updateCV("")
	post cv = "";


	-- returns all users that have a similar name to the given one ordered by a rank (distance from the user)
	-- it is guaranteed by the post condition that there are no repetitions
	public userSearchByName: seq of char ==> seq of User
	userSearchByName(name) == (
		-- the pair's key is the score and the value is the user
		dcl sortedSearch: seq of Pair := rankAndSort(Linkedin`getInstance().searchByName(name) \ {self});
		dcl result: seq of User := [];

		for pair in sortedSearch do
			result := result ^ [pair.value];

		return result
	)
	pre self in set Linkedin`getInstance().users and
			len name > 0
	post card {user | user in seq RESULT & user in set Linkedin`getInstance().users} = len RESULT;

	-- rank a search by the distance (the first element is a better match than the last one)
	private rankAndSort: set of User ==> seq of Pair
	rankAndSort(search) == (
		-- the pair's key is the score and the value is the user
		dcl sortedSearch: seq of Pair := [];
		dcl sortedSearch_temp: seq of Pair := [];
		dcl inserted: bool := false;
		dcl distance: nat1;

		for all match in set search do (
			inserted := false;
			distance := Linkedin`getInstance().distance(self, match);
			sortedSearch_temp := [];

			-- insert in the correct place
			for pair in sortedSearch do (
				if not inserted and distance < pair.key then (
					inserted := true;
					sortedSearch_temp := sortedSearch_temp ^ [mk_Pair(distance, match)];
				);
				sortedSearch_temp := sortedSearch_temp ^ [pair];
			);

			-- if it is the most distant user then it was not inserted before
```

```
                    if not inserted then
                        sortedSearch_temp := sortedSearch_temp ^
[mk_Pair(distance, match)];

                    sortedSearch := sortedSearch_temp;
                );

                return sortedSearch
        )
        pre card {user | user in set search & user in set
Linkedin`getInstance().users and user <> self} = card search;


        public msgGroup: seq1 of char * seq1 of char ==> ()
        msgGroup(groupName, msg) == (
                dcl groups: map seq1 of char to Group := Linkedin`getInstance().groups;
                groups(groupName).sendMsg(self, msg);
        )
        pre self in set Linkedin`getInstance().users and
                    groupName in set dom Linkedin`getInstance().groups;


        public pure getGroupMsgs: seq1 of char ==> seq of Group`Message
        getGroupMsgs(groupName) == return
Linkedin`getInstance().groups(groupName).checkMsgs(self)
        pre self in set Linkedin`getInstance().users and
                    groupName in set dom Linkedin`getInstance().groups;
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end User
```

User	15	100.0%	156
addConnection	27	0.0%	0
deleteCV	66	100.0%	1
getGroupMsgs	132	100.0%	1
msgGroup	123	100.0%	1
rankAndSort	89	100.0%	1
similarName	33	100.0%	45
updateCV	59	100.0%	2
userSearchByName	73	100.0%	1
User.vdmpp		92.8%	208

4. Validação do Modelo

Os testes de seguida apresentados foram executados com sucesso na totalidade:

4.1. Classe MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

operations

	-- Simulates assertion checking by reducing it to pre-condition checking.
	-- If 'arg' does not hold, a pre-condition violation will be signaled.
	protected assertTrue: bool ==> ()
	assertTrue(arg) ==
		return
	pre arg;

	-- Simulates assertion checking by reducing it to post-condition checking.
	-- If values are not equal, prints a message in the console and generates
	-- a post-conditions violation.
	protected assertEqual: ? * ? ==> ()
	assertEqual(expected, actual) ==
		if expected <> actual then (
			IO`print("Actual value (");
			IO`print(actual);
			IO`print(") different from expected (");
			IO`print(expected);
			IO`println(")\n")
		)
	post expected = actual

end MyTestCase
```

4.2. Classe LinkedinTest

```
class LinkedinTest is subclass of MyTestCase
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
	user1: User := new User("antonio");
	user2: User := new User("duarte");
	user3: User := new User("marina");
	user4: User := new User("ramadas");
	user5: User := new User("pinto");
	user6: User := new User("camilo");
	user7: User := new User("ardonio");
	user8: User := new User("cc");
	user9: User := new User("coimbras");
operations
	-- this function asserts if everything is working as it should
	-- uncomment tests supposed to fail to check if pre and post conditions are
working
	--as they should (the test are between multiline comments (/* */)
	public static main: () ==> ()
	main() ==
	(
		dcl linkedinTest: LinkedinTest := new LinkedinTest();

		IO`print("testAddUsers -> ");
		linkedinTest.testAddUsers();
		IO`println("Success");

		IO`print("testUsersName -> ");
		linkedinTest.testUsersName();
		IO`println("Success");

		IO`print("testUserSearch -> ");
		linkedinTest.testUserSearch();
		IO`println("Success");

		IO`print("testUserConnections -> ");
		linkedinTest.testUserConnections();
		IO`println("Success");

		IO`print("testSearchConnections -> ");
		linkedinTest.testSearchConnections();
		IO`println("Success");

		IO`print("testMostPopularUser -> ");
		linkedinTest.testMostPopularUser();
```

```
            IO`println("Success");

            IO`print("testUserCv -> ");
            linkedinTest.testUserCv();
            IO`println("Success");

            IO`print("testAddGroups -> ");
            linkedinTest.testAddGroups();
            IO`println("Success");

            IO`print("testAddGroupUsers -> ");
            linkedinTest.testAddGroupUsers();
            IO`println("Success");

            IO`print("testRemoveGroupUsers -> ");
            linkedinTest.testRemoveGroupUsers();
            IO`println("Success");

            IO`print("testMsgGroup -> ");
            linkedinTest.testMsgGroup();
            IO`println("Success");

            IO`print("testPopularGroup -> ");
            linkedinTest.testPopularGroup();
            IO`println("Success");

            IO`print("testGroupNames -> ");
            linkedinTest.testGroupNames();
            IO`println("Success");

            IO`print("testGroupInfo -> ");
            linkedinTest.testGroupInfo();
            IO`println("Success");

            IO`print("testUsersDistance -> ");
            linkedinTest.testUsersDistance();
            IO`println("Success");

            IO`print("testUsersAvgDistance -> ");
            linkedinTest.testUsersAvgDistance();
            IO`println("Success");

            IO`print("testUserAvgDistance -> ");
            linkedinTest.testUserAvgDistance();
            IO`println("Success");
        );

        -- test if the registration of users in the social network is working
correctly
        private testAddUsers: () ==> ()
        testAddUsers() == (
```

```
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		assertEqual(9, card linkedin.users);

		assertEqual({user1, user2, user3,	user4, user5,	user6,
user7,	user8, user9}, linkedin.users);

		-- this test is supposed to fail (the same user cannot register twice
in the network)
		/**linkedin.addUser(user1);**/
	);

	-- test if the return of the users' name in the social network is working
correctly
	private testUsersName: () ==> ()
	testUsersName() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		assertEqual([user1.name,	user2.name,	user3.name,	user4.name,
user5.name,	user6.name,	user7.name,	user8.name,	user9.name],
linkedin.getUsersName());
	);

	-- test user search by their name
	private testUserSearch: () ==> ()
	testUserSearch() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		-- there is not a user whose name match this search
		assertEqual({}, linkedin.searchByName("nil user"));

		-- match at the beginning of the name
		assertEqual({user1}, linkedin.searchByName("ant"));

		-- match at the end of the name
		assertEqual({user2}, linkedin.searchByName("te"));

		-- multiple results
		assertEqual({user1, user7}, linkedin.searchByName("onio"));

		-- the first 3 users connect to each other
		linkedin.addConnection(user1, user2);
		linkedin.addConnection(user1, user3);
		linkedin.addConnection(user2, user3);
		-- 3 is connected with 4, 4 with 7 and 7 with 6
```

```
		linkedin.addConnection(user3, user4);
		linkedin.addConnection(user4, user7);
		linkedin.addConnection(user7, user6);

		-- the search is sorted according to the distance of the user
		assertEqual([user2,	user3,	user4,	user7,	user6,	user9],
user1.userSearchByName("a"));
	);

	-- test the creation of connections between users
	private testUserConnections: () ==> ()
	testUserConnections() == (
		dcl dummyUser: User := new User("dummy user");
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		-- add a new connection between users
		linkedin.addConnection(user1, user2);
		assertEqual({user2}, user1.connections);
		assertEqual({user1}, user2.connections);

		-- this is suppose to fail (cannot add a connection between an user in
the network and one that is not)
		/*linkedin.addConnection(dummyUser, user2);*/
	);

	-- test the search of commun connections between users
	private testSearchConnections: () ==> ()
	testSearchConnections() == (
		dcl dummyUser: User := new User("dummy user");
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		-- the first 3 users connect to each other
		linkedin.addConnection(user1, user2);
		linkedin.addConnection(user1, user3);
		linkedin.addConnection(user2, user3);

		-- the commun connection is only 1 user
		assertEqual({user3}, linkedin.findConnections(user1, user2));

		-- this assert is suppose to fail (dummyUser is not in the network)
		/*assertEqual({}, linkedin.findConnections(dummyUser, user2));*/
	);

	-- test the search for the most famous user
	private testMostPopularUser: () ==> ()
	testMostPopularUser() == (
		dcl dummyUser: User := new User("dummy user");
```

```
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		linkedin.addConnection(user1, user2);
		linkedin.addConnection(user1, user3);
		linkedin.addConnection(user2, user3);
		linkedin.addConnection(user1, user4);

		assertEqual(user1, linkedin.mostFamousUser());
	);

	-- test the manipulation of users' cv
	private testUserCv: () ==> ()
	testUserCv() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		user1.updateCV("---User1 CV---Welcome to my CV!!");
		assertEqual("---User1 CV---Welcome to my CV!!", user1.cv);

		user1.deleteCV();
		assertEqual("", user1.cv);
	);

	-- test the creation of groups
	private testAddGroups: () ==> ()
	testAddGroups() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		linkedin.addGroup("Antonio's Group", user1);
		assertEqual(1, card dom linkedin.groups);

		-- this test is suppose to fail (cannot be two groups with the same
name)
		/*linkedin.addGroup("Antonio's Group");*/
	);

	-- test the registration of users in groups
	private testAddGroupUsers: () ==> ()
	testAddGroupUsers() == (
		dcl dummyUser: User := new User("dummy user");
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		linkedin.addGroup("Antonio's Group", user1);
```

```
            linkedin.addGroupUser("Antonio's Group", user2);
            linkedin.addGroupUser("Antonio's Group", user3);

            assertEqual({user1,    user2,    user3},    linkedin.groups("Antonio's
Group").users);

            -- this test is suppose to fail (the user is not in the network)
            /*linkedin.addGroupUser("Antonio's Group", dummyUser);*/

            -- this test is suppose to fail (the group does not exist)
            /*linkedin.addGroupUser("Dummy's Group", user1);*/
      );

      -- test the removal of users from groups
      private testRemoveGroupUsers: () ==> ()
      testRemoveGroupUsers() == (
            dcl linkedin: Linkedin := Linkedin`clearInstance();

            createAndAddUsers();

            linkedin.addGroup("Antonio's Group", user1);

            linkedin.addGroupUser("Antonio's Group", user2);

            linkedin.removeGroupUser("Antonio's Group", user2);
            assertEqual({user1}, linkedin.groups("Antonio's Group").users);

            -- this test is suppose to fail (cannot remove a user that is not in
the group)
            /*linkedin.removeGroupUser("Antonio's Group", user2);*/

            -- this test is suppose to fail (cannot remove a user from a group that
does not exist)
            /*linkedin.removeGroupUser("Dummy's Group", user1);*/
      );

      -- test messaging the group
      private testMsgGroup: () ==> ()
      testMsgGroup() == (
            dcl linkedin: Linkedin := Linkedin`clearInstance();

            createAndAddUsers();

            linkedin.addGroup("Antonio's Group", user1);

            user1.msgGroup("Antonio's Group", "Hello, World!");
            assertEqual(["Hello, World!"], user1.getGroupMsgs("Antonio's Group"));

            -- this test is suppose to fail (the user does not have access to the
group)
            /*linkedin.getGroupMsgs("Antonio's Group", user2);*/
```

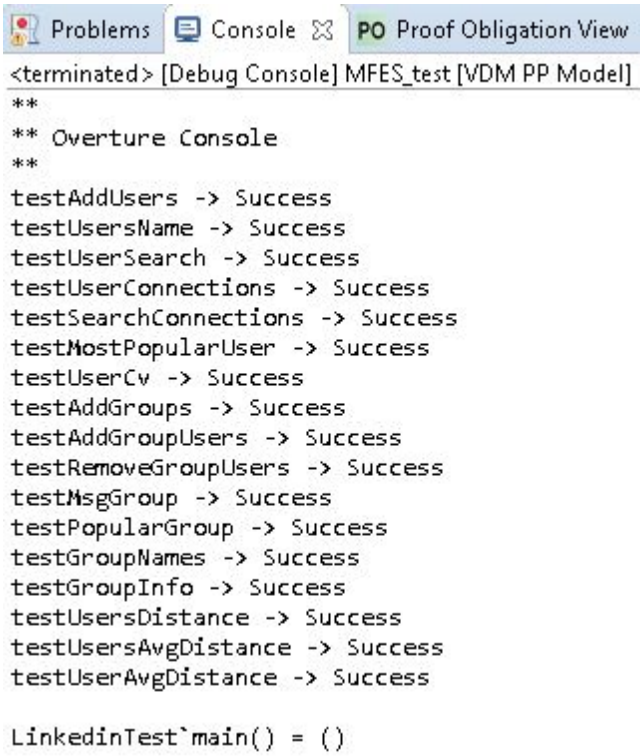
```
		-- this test is suppose to fail (the user is not in the group)
		/*linkedin.msgGroup("Antonio's Group", user2, "Dummy Message!");*/

		-- this test is suppose to fail (there is no such group)
		/*linkedin.msgGroup("Dummy's Group", user1, "Dummy Message!");*/
	);

	-- test the search for the group with more registrated users
	private testPopularGroup: () ==> ()
	testPopularGroup() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		linkedin.addGroup("Antonio's Group", user1);
		linkedin.addGroupUser("Antonio's Group", user2);
		linkedin.addGroupUser("Antonio's Group", user3);

		linkedin.addGroup("Duarte's Group", user1);
		linkedin.addGroupUser("Duarte's Group", user2);

		linkedin.addGroup("Marina's Group", user1);

		assertEqual("Antonio's Group", linkedin.mostPopularGroup());
	);

	-- test to check if all the groups' name is returned
	private testGroupNames: () ==> ()
	testGroupNames() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		linkedin.addGroup("Antonio's Group", user1);
		linkedin.addGroup("Duarte's Group", user1);
		linkedin.addGroup("Marina's Group", user1);

		assertEqual({"Antonio's Group", "Duarte's Group", "Marina's Group"},
linkedin.getGroupNames());
	);

	-- test if the returned information of a given group is correct
	private testGroupInfo: () ==> ()
	testGroupInfo() == (
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		createAndAddUsers();

		linkedin.addGroup("Antonio's Group", user1);
```

```
            assertEqual(linkedin.groups("Antonio's                          Group"),
linkedin.getGroupInfo("Antonio's Group"));

            -- this test is supposed to fail (there is not such group)
            /*linkedin.getGroupInfo("Dummy's Group");*/
      );

      -- test the distance between users
      private testUsersDistance: () ==> ()
      testUsersDistance() == (
            dcl dummyUser: User := new User("dummy user");
            dcl linkedin: Linkedin := Linkedin`clearInstance();

            createAndAddUsers();

            linkedin.addConnection(user1, user2);
            linkedin.addConnection(user1, user3);

            -- the distance to the user itself is zero
            assertEqual(0, linkedin.distance(user1, user1));

            -- the distance between two users connected is 1
            assertEqual(1, linkedin.distance(user1, user2));

            -- the distance between users is bilateral
            assertEqual(linkedin.distance(user2,  user1),  linkedin.distance(user1,
user2));

            -- the users are not directly connected
            assertEqual(2, linkedin.distance(user2, user3));

            -- the users are not connected
            assertEqual(Linkedin`INFINITY, linkedin.distance(user5, user6));

            -- this test is supposed to fail (dummyUser is not in the network)
            /*assertEqual("ERROR", linkedin.distance(dummyUser, user3));*/
      );

      -- test to check if it is made the correct average of the distance between
users
      private testUsersAvgDistance: () ==> ()
      testUsersAvgDistance() == (
            dcl linkedin: Linkedin := Linkedin`clearInstance();

            user1 := new User("antonio");
            user2 := new User("duarte");
            user3 := new User("marina");
            linkedin.addUser(user1);
            linkedin.addUser(user2);
            linkedin.addUser(user3);
```

```
		linkedin.addConnection(user1, user2);
		linkedin.addConnection(user1, user3);
		linkedin.addConnection(user2, user3);

		assertEqual(1, linkedin.averageUserDistance());
	);

	-- test to check if it is made the correct average of the distance between an
user to all users
	private testUserAvgDistance: () ==> ()
	testUserAvgDistance() == (
		dcl dummyUser: User := new User("dummy user");
		dcl linkedin: Linkedin := Linkedin`clearInstance();

		user1 := new User("antonio");
		user2 := new User("duarte");
		user3 := new User("marina");
		linkedin.addUser(user1);
		linkedin.addUser(user2);
		linkedin.addUser(user3);

		linkedin.addConnection(user1, user2);
		linkedin.addConnection(user1, user3);

		assertEqual(1.5, linkedin.averageDistanceToOthers(user3));

		-- this test is supposed to fail (dummyUser is not in the network)
		/*linkedin.averageDistanceToOthers(dummyUser));*/
	);

	private createAndAddUsers: () ==> ()
	createAndAddUsers() ==     (
		user1 := new User("antonio");
		user2 := new User("duarte");
		user3 := new User("marina");
		user4 := new User("ramadas");
		user5 := new User("pinto");
		user6 := new User("camilo");
		user7 := new User("ardonio");
		user8 := new User("cc");
		user9 := new User("coimbras");

		Linkedin`getInstance().addUser(user1);
		Linkedin`getInstance().addUser(user2);
		Linkedin`getInstance().addUser(user3);
		Linkedin`getInstance().addUser(user4);
		Linkedin`getInstance().addUser(user5);
		Linkedin`getInstance().addUser(user6);
		Linkedin`getInstance().addUser(user7);
		Linkedin`getInstance().addUser(user8);
		Linkedin`getInstance().addUser(user9);
```

```
      );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end LinkedinTest
```

4.3. Resultados



Teste	Descrição
testAddUsers	Verifica se o requisito **R01** funciona correctamente.
testUsersName	Verifica que é possível ir buscar o utilizador pelo nome.
testUserSearch	Verifica se o requisito **R09** funciona correctamente.
testUserConnections	Verifica se o requisito **R03** funciona correctamente.
testSearchConnections	Verifica se o requisito **R04** funciona correctamente.
testMostPopularUser	Verifica se o requisito **R08** funciona correctamente.
testUserCv	Verifica se o requisito **R02** funciona correctamente.
testAddGroups	Verifica se o requisito **R10** funciona correctamente.

testAddGroupUsers	Verifica se a parte do requisito **R13** de adição de utilizadores em grupos funciona correctamente.
testRemoveGroupUsers	Verifica se a parte do requisito **R13** de remoção de utilizadores de grupos funciona correctamente.
testMsgGroup	Verifica se os requisitos **R14 e R15** funcionam correctamente.
testPopularGroup	Verifica se o requisito **R16** funciona correctamente.
testGroupNames	Verifica se o requisito **R11** funciona correctamente.
testGroupInfo	Verifica se o requisito **R12** funciona correctamente.
testUsersDistance	Verifica se o requisito **R05** funciona correctamente.
testUsersAvgDistance	Verifica se o requisito **R07** funciona correctamente.
testUserAvgDistance	Verifica se o requisito **R06** funciona correctamente.

5. Verificação do modelo

5.1. Exemplo de uma verificação de domínio

Uma *proof obligation* gerada pelo Overture foi:

No.	PO Name	Type
47	Linkedin`getGroupInfo	legal map application

O código em análise (com as partes relevantes da map application sublinhadas) é:

```
-- returns all the information about a group
public pure getGroupInfo: seq1 of char ==> Group
getGroupInfo(name) == return groups(name)
pre name in set dom groups;
```

Neste caso podemos provar facilmente utilizando a precondição `'name in set dom groups'` que assegura que o *map* é acedido apenas dentro do seu domínio.

5.2. Exemplo de uma verificação de invariante

Uma outra *proof obligations* gerada pelo Overture é:

No.	PO Name	Type
35	Linkedin`addConnection(User, User)	state invariant holds

O código em análise (com as partes relevantes da map application sublinhadas) é:

```
-- add a new connections between users
public addConnection: User * User ==> ()
addConnection(user1, user2) == (
--|| (user1.addConnection(user2), user2.addConnection(user1))
  dcl user1_temp: User := user1;
  dcl user2_temp: User := user2;
  atomic (
       user1_temp.connections := user1.connections union {user2};
       user2_temp.connections := user2.connections union {user1}
  );
)

pre
   -- they are not the same user
   user1 <> user2 and
   -- both users exist in the database
   user1 in set users and
   user2 in set users and
   -- there's no connection between them
   user2 not in set user1.connections and
   user1 not in set user2.connections
post user2 in set user1.connections and user1 in set user2.connections;
```

A invariante relevante sobre análise é:

```
inv forall user1 in set users & forall user2 in set user1.connections &
       user1 in set user2.connections;
```

Depois da execução do bloco '`atomic`' nós temos o seguinte bloco de código que adiciona uma ligação entre dois utilizadores (é adicionado o outro utilizador nas conexões):

```
user1_temp.connections := user1.connections union {user2}
       and user2_temp.connections := user2.connections union {user1}
```

Temos de provar que após a execução do bloco a invariante se mantém, ou seja, que as conexões entre utilizadores são bidirecionais (se um utilizador tem outro como conexão, então a conexão inversa existe no outro utilizador):

```
user1_temp.connections := user1.connections union {user2}
       and user2_temp.connections := user2.connections union {user1} =>
forall user1 in set users & forall user2 in set user1.connections &
       user1 in set user2.connections;
```

O que obviamente é verdadeiro.

6. Geração de código Java

A geração de código Java a partir do código feito em VDM++ é bastante simples e intuitiva não sendo necessário fazer nenhuma alteração. Assim, para correr o projeto com a *Command Line Interface* (CLI) basta copiar a classe do tópico 6.1 para um ficheiro ao mesmo nível que os ficheiros Java gerados, mudar o nome do *package* caso necessário, compilar e executar.

6.1. Command Line Interface

```
package MFES;

import java.io.IOException;
import java.util.*;

import org.overture.codegen.runtime.VDMSeq;
import org.overture.codegen.runtime.VDMSet;

public class CommandLineInterface{
	public static final String BACK_INPUT = "b";
	public static final String MENU_INPUT = "m";

	Linkedin linkedIn = Linkedin.getInstance();
	Scanner scanner = new Scanner(System.in);

	public CommandLineInterface(){}

	public static void main(String[] args){
		CommandLineInterface cli = new CommandLineInterface();

		cli.mainMenu();
	}


	public void mainMenu(){
		System.out.println("Welcome to LinkedIn!!!");
		printDivision("Main Menu");

		ArrayList<String> options = new ArrayList<>();
		options.add("Users");
		options.add("Groups");
		options.add("Search users by name");
		options.add("Stats");

		String input = printOptions(options, false);
```

```
		switch(input){
			case "Users":
				usersMenu(true);
				mainMenu();
				break;
			case "Groups":
				groupsMenu(true);
				mainMenu();
				break;
			case "Search users by name":
				searchUsersByName(true);
				mainMenu();
				break;
			case "Stats":
				statsMenu(true);
				mainMenu();
				break;
			case BACK_INPUT:
				break;
			case MENU_INPUT:
				break;
		}
	}

	private void statsMenu(boolean hasParent) {
		printDivision("Stats");

		ArrayList<String> options = new ArrayList<>();
		options.add("Most famous user");
		options.add("Most famous group");
		options.add("Average User Distance");

		String input = printOptions(options, hasParent);
		switch(input){
			case "Most famous user":
				mostFamousUser(true);
				statsMenu(hasParent);
				break;
			case "Most famous group":
				mostFamousGroup(true);
				statsMenu(hasParent);
				break;
			case "Average User Distance":
				averageUsersDistance(true);
				statsMenu(hasParent);
				break;
			case BACK_INPUT:
				return;
			case MENU_INPUT:
				mainMenu();
				break;
```

```
		}

	}

	private void averageUsersDistance(boolean hasParent) {
		System.out.println();
		System.out.print("Average user info: ");
		Number avgUserDist = linkedIn.averageUserDistance();

		System.out.println(""+avgUserDist);

		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	private void mostFamousGroup(boolean hasParent) {
		System.out.println();
		System.out.print("Most famous group: ");
		String group= linkedIn.mostPopularGroup();

		System.out.println(group);

		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

	}

	private void mostFamousUser(boolean hasParent) {
		System.out.println();
		System.out.println("Most famous user: ");
		User user = linkedIn.mostFamousUser();

		System.out.println("Name: "+user.name);
		System.out.println("Id: "+user.id);
		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

	}

	private void searchUsersByName(boolean hasParent) {
		printDivision("Search");
		System.out.println("Search: ");
```

```
            String search = scanner.nextLine();

            Set<User> searchResult = linkedIn.searchByName(search);

            printSearcResult(searchResult);

            ArrayList<String> options = new ArrayList<>();
            options.add("Go to user profile");

            String input = printOptions(options, hasParent);
            switch(input){
                  case "Go to user profile":
                        User user = enterUserId();
                        userProfile(true, user);
                        searchUsersByName(hasParent);
                        break;
                  case BACK_INPUT:
                        return;
                  case MENU_INPUT:
                        mainMenu();
                        break;
            }



      }

      private void printSearcResult(Set<User> searchResult) {
            System.out.println("Results:");
            printListUsers(searchResult);
            System.out.println();
      }

      private void groupsMenu(boolean hasParent) {
            printDivision("List of Groups");

            ArrayList<String> options = new ArrayList<>();
            options.add("See group's info");
            options.add("Add a new group");

            String input = printOptions(options, hasParent);
            switch(input){
                  case "See group's info":
                        String groupName = selectGroup(hasParent);
                        groupsInfo(true, groupName);
                        groupsMenu(hasParent);
                        break;
                  case "Add a new group":
                        addNewGroup(true);
                        groupsMenu(hasParent);
                        break;
```

```
                case BACK_INPUT:
                        return;
                case MENU_INPUT:
                        mainMenu();
                        break;
        }

}

private void groupsInfo(boolean hasParent, String groupName) {

        Group group;
        try{
                group = linkedIn.getGroupInfo(groupName);
        }catch(Exception e){
                System.out.println("Invalid group name");
                groupsInfo(hasParent, selectGroup(hasParent));
                return;
        }
        printDivision("Group's Info");

        printDivision(groupName);
        System.out.println();
        System.out.println("Users:");
        printListUsers(group.users);
        System.out.println();

        ArrayList<String> options = new ArrayList<>();
        options.add("Add existing user to group");
        options.add("Remove user from group");

        String input = printOptions(options, hasParent);
        switch(input){
                case "Add existing user to group":
                        addUserToGroup(true, groupName);
                        groupsInfo(hasParent, groupName);
                        break;
                case "Remove user from group":
                        removeUserFromGroup(true, groupName);
                        groupsInfo(hasParent, groupName);
                        break;
                case BACK_INPUT:
                        return;
                case MENU_INPUT:
                        mainMenu();
                        break;
        }
}

private void removeUserFromGroup(boolean hasParent, String group) {
        User user = enterUserId();
```

```
		linkedIn.removeGroupUser(group , user);
	}

	private void addUserToGroup(boolean hasParent, String group) {
		User user = enterUserId();

		linkedIn.addGroupUser(group , user);
	}

	private User enterUserId(){
		System.out.print("Enter user's id: ");
		String id = scanner.nextLine();

		while(!isValidUserId(id)){
			printInvalidUsersId();
			System.out.print("Enter user's id: ");
			id = scanner.nextLine();
		}

		return getUserById(id);
	}

	private User getUserById(String idStr) {

		int id;
		try{
			id= Integer.parseInt(idStr);
		}catch(Exception e){
			return null;
		}
		Set<User> users = linkedIn.users;
		for(User user : users){
			if(id == user.id.intValue()){
				return user;
			}
		}
		return null;
	}

	private boolean isValidUserId(String idStr) {

		try{
			int id = Integer.parseInt(idStr);

			Set<User> users = linkedIn.users;


			for(User user : users){

				//System.out.println("LOG: Avaliar ids: user_"+user.id +
```

```
" e id_"+id);

                        if(user.id.intValue() == id){
                                return true;
                        }
                }

                //System.out.println("LOG: id não válido" );

                return false;
        }catch(Exception e){
                //e.printStackTrace();
                return false;
        }

    }

    private String selectGroup(boolean hasParent) {
        System.out.println();
        System.out.print("Enter group's name: ");
        String name = scanner.nextLine();

        return name;
    }

    private void addNewGroup(boolean hasParent) {
        User user = enterUserId();

        System.out.print("Enter group's name: ");
        String group = scanner.nextLine();

        linkedIn.addGroup(group, user);
    }

    private void printUserInfo(User user){
        System.out.println("ID: "+user.id);
        System.out.println("Name: "+user.name);
        System.out.println("CV: \n"+user.cv);
        System.out.println();
        System.out.println("Connections: ");
        printListUsers(user.connections);
        System.out.println();
        System.out.println();
    }

    public void usersMenu(boolean hasParent){
        printDivision("List of users");
        System.out.println();
        printListUsers();
        System.out.println();
```

```
            ArrayList<String> options = new ArrayList<>();
            options.add("See User's Profile");
            options.add("Add User");
            options.add("Add a new connection between two users");
            options.add("Distance between two users");
            options.add("Average distance of an user to other users");

            String input = printOptions(options, hasParent);
            switch(input){
                  case "See User's Profile":
                        User user = enterUserId();
                        userProfile(true, user);
                        usersMenu(true);
                        break;
                  case "Add User":
                        addUser(true);
                        usersMenu(true);
                        break;
                  case "Add a new connection between two users":
                        addConnectionUsers(true);
                        usersMenu(true);
                        break;
                  case "Distance between two users":
                        distanceBetweenTwoUsers(true);
                        usersMenu(true);
                        break;
                  case "Average distance of an user to other users":
                        averageDistanceToUser(true);
                        usersMenu(true);
                        break;
                  case BACK_INPUT:
                        return;
                  case MENU_INPUT:
                        mainMenu();
                        break;

            }
      }

      private void averageDistanceToUser(boolean hasParent) {
            User user = enterUserId();
            System.out.println("Average distance to user "+user.name+":
"+linkedIn.averageDistanceToOthers(user));
      }

      private void distanceBetweenTwoUsers(boolean hasParent) {
            System.out.print("User 1.");
            User user1 = enterUserId();

            System.out.print("User 2.");
```

```
                User user2 = enterUserId();

                Number distance = linkedIn.distance(user1, user2);

                System.out.println("Distance between the user "+user1.name+" and user
"+user2.name+": "+distance);

                try {
                        Thread.sleep(5000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }

        }

        private void addConnectionUsers(boolean hasParent) {
                System.out.print("User 1.");
                User user1 = enterUserId();

                System.out.print("User 2.");

                User user2 = enterUserId();

                linkedIn.addConnection(user1, user2);
                System.out.println("Connection Added");

                try {
                        Thread.sleep(5000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        private void addUser(boolean hasParent) {
                System.out.print("Enter new user's name: ");
                String username = scanner.nextLine();

                User user = new User(username);

                linkedIn.addUser(user);
        }

        private void userProfile(boolean hasParent, User user) {


                printDivision("User");
                printUserInfo(user);

                ArrayList<String> options = new ArrayList<>();
                options.add("Add connection to other users");
                options.add("Update CV");
```

```
        options.add("Delete CV");
        options.add("Send a message to a group");
        options.add("Check messages from a group");

        String input = printOptions(options, hasParent);
        switch(input){
                case "Add connection to other users":
                        addConnectionToUser(true, user);
                        userProfile(true, user);
                        break;
                case "Update CV":
                        updateCv(true, user);
                        userProfile(true, user);
                        break;
                case "Delete CV":
                        deleteCV(true, user);
                        userProfile(true, user);
                        break;
                case "Send a message to a group":
                        sendMessageToGroup(true, user);
                        userProfile(true, user);
                        break;
                case "Check messages from a group":
                        checkMessagesFromGroup(true, user);
                        userProfile(true, user);
                        break;
                case BACK_INPUT:
                        return;
                case MENU_INPUT:
                        mainMenu();
                        break;
        }

}

private void checkMessagesFromGroup(boolean hasParent, User user) {
        System.out.print("Enter group's name: ");
        String input = scanner.nextLine();
        try{
                printDivision("Messages");
                VDMSeq seq = user.getGroupMsgs(input);

                printGroupMessages(seq);

                ArrayList<String> options = new ArrayList<>();


                input = printOptions(options, hasParent);
                switch(input){
                        case BACK_INPUT:
                                return;
```

```
                        case MENU_INPUT:
                                mainMenu();
                                break;
                        }
                }catch(Exception e){
                        System.out.println("Invalid group name");
                        checkMessagesFromGroup(hasParent, user);
                }

        }

        private void printGroupMessages(VDMSeq seq){
                for(int i = 0; i < seq.size(); i++){
                        System.out.println(i+"- "+ seq.get(i));
                        System.out.println();

                }
                System.out.println();
                System.out.println();
        }

        private void deleteCV(boolean hasParent, User user) {
                boolean delete = confirmationDialog();

                if(delete){
                        user.deleteCV();
                }else{
                        return;
                }
        }

        private boolean confirmationDialog(){
                System.out.println("Are you sure? (Y\\N)");
                String input = scanner.nextLine().toLowerCase();
                boolean validInput = false;


                while(!validInput){
                        switch(input){
                                case "y":
                                        validInput = true;
                                        return true;
                                case "n":
                                        validInput = true;
                                        return false;
                                default:
                                        printInvalidInputMessage();
                                        input = scanner.nextLine().toLowerCase();
                                        break;
                        }
                }
```

```
            return false;

        }

        private void sendMessageToGroup(boolean hasParent, User user) {
            System.out.print("Enter group's name: ");
            String group = scanner.nextLine();
            System.out.println("Enter message:");
            String input = scanner.nextLine();

            try{

                user.msgGroup(group, input);


            }catch(Exception e){
                System.out.println("Invalid group name");
                checkMessagesFromGroup(hasParent, user);
            }

        }

        private void updateCv(boolean hasParent, User user) {
            System.out.println("Edit CV:");
            String input = scanner.nextLine();
            user.updateCV(input);
        }

        private void addConnectionToUser(boolean hasParent, User user) {
            User user2 = enterUserId();
            user.addConnection(user2);

            System.out.println("Connection added!");
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        }

        private void printListUsers() {
            Set<User> users = linkedIn.users;
            printListUsers(users);

        }

        private void printListUsers(Set<User> users){
            for(User user : users){
                System.out.println("Name:
```

```
"+user.name+spacesFill(16,user.name.length())+"ID: "+user.id);
		}
	}

	public void printDivision(String menu){
		System.out.println();

System.out.println("------------------------"+menu+"-----------------------");

	}

	public String printOptions(ArrayList<String> options, boolean hasParent){
		for(int i = 0; i < options.size(); i++){
			System.out.println((i+1) +"- "+options.get(i));

		}

		if(hasParent){
			System.out.println("'b' to go back or 'm' to go to the main
menu.");
		}
		System.out.print("Select one of the options: ");

		boolean validInput = false;
		String input = "";
		while(!validInput){
			input = scanner.nextLine();
			validInput = isValidInput(options, hasParent, input);
			if(!validInput){
				printInvalidInputMessage();
			}
		}


		if(input.equals(BACK_INPUT) || input.equals(MENU_INPUT)){
			return input;
		}else{
			return options.get(Integer.parseInt(input)-1);
		}
	}

	public boolean isValidInput(ArrayList<String> options, boolean hasParent,
String input) {
		if(hasParent){
			if(input.equals(BACK_INPUT) || input.equals(MENU_INPUT)){
				return true;
			}
		}
		try{
			int i = Integer.parseInt(input);
```

```
                if(i > 0 && i <= options.size()){
                        return true;
                }else{
                        return false;
                }
        }catch(Exception e){
                return false;
        }
    }

    public void printInvalidInputMessage() {
        System.out.println("Invalid input!!!");

    }

    public void printInvalidUsersId(){
        System.out.println("Invalid user's id");
    }

    public String spacesFill(int maxSize, int used){
        String retStr = "";
        int iMax= maxSize-used;

        if (iMax <= 0){
                return " ";
        }

        for(int i = 0; i < iMax; i++){
                retStr += " ";
        }

        return retStr;
    }

}
```

7. Conclusão

De acordo com os requisitos identificados no enunciado foram todos os objetivos foram concretizados com sucesso. Como melhoramento propusemo-nos a implementar grupos, implementados com sucesso também. No entanto, existe sempre espaço para melhorias, como por exemplo, os utilizadores podiam realizar publicações dando a possibilidade aos utilizadores de colocarem “Like”. Outra das possíveis melhorias seria implementar uma gestão de notificações que fosse avisando o utilizador de novas conexões e publicações.

Todos trabalhamos em conjunto e em sintonia o que permitiu acelerar o desenvolvimento do trabalho. Assim, todos os elementos da equipa participaram equitativamente:
33% - António
33% - Duarte
33% - Marina

Em suma, este projeto elucidou-nos sobre o impacto de uma boa implementação e de como existem ferramentas para verificar a sua correção. Em sistemas críticos esta verificação revela-se essencial. Assim, acreditamos que esta área seja muito importante em Engenharia Informática. Resta agradecer aos professores desta unidade curricular por todo o apoio prestado.

8. Referências

1. Apontamentos da unidade curricular Métodos Formais de Engenharia de Software
2. Overture Quick Start Exercise, https://docs.google.com/document/d/1cf1cn2qbELCMaHZcBut__0dQL9HGnHlk-I7dmM5q5kw/pub
3. Overture tool web site, http://overturetool.org
4. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
5. Linkedin, https://pt.linkedin.com/